A New High Radix-2r ($r \geq 8$) Multibit Recoding Algorithm for Large Operand Size ($N \geq 32$) Multipliers.

Abdelkrim K. Oudjida, Nicolas Chaillet, Mohamed L. Berrandjia, Ahmed Liacha

A New High Radix-2^r ($r \geq 8$) Multibit Recoding Algorithm for Large Operand Size ($N \geq 32$) Multipliers

A.K. Oudjida[1], N. Chaillet[2], M.L. Berrandjia[1], and A. Liacha[1]

(1) Centre de Développement des Technologies Avancées, Algiers, Algeria

(2) Institut FEMTO-ST, Besançon, France

***Abstract*—This paper addresses the problem of multiplication with large operand sizes ($N \geq 32$). We propose a new recursive recoding algorithm that shortens the critical path of the multiplier and reduces the hardware complexity of partial-product-generators as well. The new recoding algorithm provides an optimal space/time partitioning of the multiplier architecture for any size N of the operands. As a result, the critical path is drastically reduced to $3\sqrt[3]{N/2} - 3$ with no area overhead in comparison to modified Booth algorithm that shows a critical path of $N/2$ in adder stages. For instance, only 7 adder stages are needed for a 64-bit two's complement multiplier. Confronted to reference algorithms for $N=64$, important gain ratios of 1.62, 1.71, 2.64 are obtained in terms of multiply-time, energy consumption per multiply-operation, and total gate count, respectively.**

***Index Terms*— High-Radix Multiplication, Low-Power Multiplication, Multibit Recoding Multiplication, Partial Product Generator (PPG), Register-Transfer-Level (RTL)**

I. Background and Motivation

In multiplication-intensive applications, as in digital signal processing or process control, multiply-time is a critical factor that limits the whole system performance. When these types of applications are embedded, energy consumption per multiply operation becomes an additional critical issue. Furthermore, in large-operand-size applications ($N \geq 32$), the need for a scalable architecture is essential to ensure a linear increase $O(N)$ of multiply-time while multiplier size grows quadratically $O(N^2)$ with operand bit-length N. Consequently, *high-speed*, *low-power*, and *highly-scalable* architecture are the three major requirements for today's general-purpose multipliers [1].

However, large operand size multipliers are very time consuming. To comply with time constraint of a given application, we need a multiplication algorithm that allows, to some extent, a parameterized reduction (N/r) of the multiply-time without sacrificing area. This is achieved if, and only if the total critical path can be properly shortened by reducing the number of partial products (PPs) and exploiting inherent parallelism. Theoretically, only the signed multibit recoding multiplication algorithm [2] is capable of such a drastic reduction (N/r) of the PP number, given that $r+1$ is the number of bits of the multiplier that are simultaneously treated ($1<r\leq N/2$). Unfortunately, this algorithm requires the pre-computation of a number of odd-multiples of the multiplicand (until $(2^{r-1}-1).X$) that scales linearly with r. The large number of odd-multiples not only requires a considerable amount of multiplexers to perform the necessary complex recoding into partial product generators (PPG), but dramatically increases the routing density as well. Therefore, a reverse effect occurs that offsets speed and power benefits of the compression factor N/r. This is the main reason why the multibit recoding algorithm was abandoned. Moreover, in industry commercial designs do not exceed $r=4$ (radix-16). A hybrid radix-4/-8 is proposed in [3] for low-power multimedia applications. To increase the speed of the multiplier, most ancient processors employed radix-8, such as: Fchip [4], IBM S/390 [5], Alpha RISC [6], IA-32 [7] and AMDK7 [8]. While radix-16 is used only in the most recent Intel processors: 64 and IA-32 [9], and Itanium-Poulson [10].

In research, the highest radix algorithms are proposed in the works of Seidel et *al*. [11] and Dimitrov et *al*. [12]. Both works rely upon advanced arithmetic to determine minimal number-bases that are representatives of the digits resulting from larger multibit recoding. The objective is to eliminate information redundancy inside $r+1$ bit-length slices for a more compact PPG. This is achievable as long as no or just very few odd- multiples are required.

Seidel introduced a secondary recoding of digits issued from an initial multibit recoding for $5\leq r\leq 16$. The recoding scheme is based on balanced complete residue system. Though it significantly reduces the number of partial products (N/r for $5\leq r\leq 16$), it requires some odd-multiples for $r \geq 8$. Dimitrov proposed a new recoding scheme based on double base number system for $6\leq r\leq 11$. The algorithm is limited to unsigned multiplication and requires larger number of odd-multiples. Both algorithms [11][12] require a PPG that includes a number of adders to accumulate intermediary partial products corresponding to recoded elementary digits.

In fact, odd-multiples are not the only problem for a compact PPG. Recoding large slices ($r \geq 8$) in a mono-bloc PPG such as in [11][12], requires the use of an RTL "case statement" with $r+1$ entries. In this case, 2^{r+1} combinations must be processed, which yields to a huge amount of multiplexer resources. Thus, mono-bloc PPG recoding is incompatible with high radix ($r \geq 8$) approach whose purpose is to reduce the multiply-time (N/r) of large operand size ($N \geq 32$) multipliers.

The objective of this paper is to overcome these two above-mentioned shortcomings. To achieve such a goal, the multibit recoding multiplication algorithm is revisited [2]. Its design space is extended by the introduction of a new recursive version that enabled to solve the hard problem of radix-2^r two's complement multiplication for any value of r. The solution consists essentially in dividing the high radix-2^r mono-bloc PPG_j (Fig. 1.a) into a number of lower sub-radix-2^s odd-multiple free PPG_{ji} (Fig. 1.b), such as s is a divider of r . As direct benefits of the partitioning of Fig. 1.b:

- there is no need to pre-compute odd-multiples of the multiplicand, which drastically reduces the required amount of hardware resources and routing;
- since the size of PPG_{ji} entry is much smaller than the size of PPG_j one ($s\leq r/2$), the total multiplexing logic required by RTL "case statements" to recode the entries is greatly reduced;

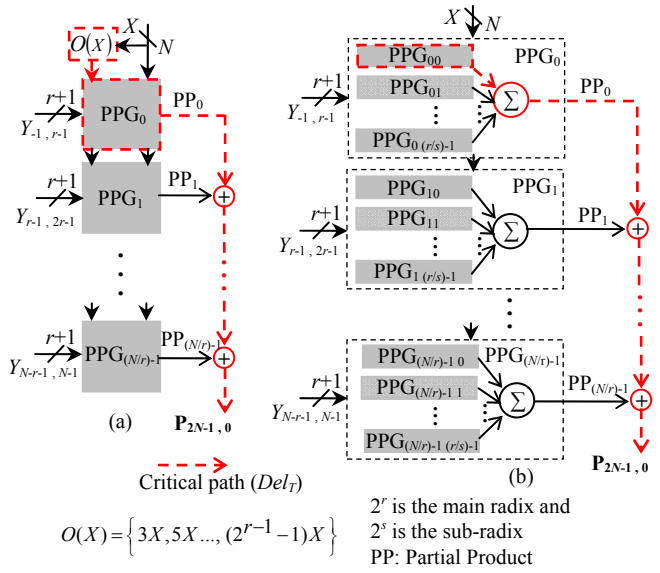

Fig. 1. Generalized $N\times N$ bit radix-2^r parallel multiplier.

(a) Critical path in conventional [2][4][5][6][7][8] and recent [3][9][10] [11][12] radix-2^r multipliers. $O(X)$ is the necessary set of odd-multiples corresponding to radix-2^r recoding. PPG_j of [11][12] includes a number of adders to accumulate intermediary partial product.

(b) Critical path in our proposed radix-2^r multipliers. Main features are: no odd-multiples, much more compact PPG_j, much shorter critical path.

- the possibility to simultaneously process larger bit slices ($r\geq 16$) radically shortens the critical path in terms of adder levels, especially for very large operand sizes (N≥64).

Guided by accurate area heuristics, the final result of an optimization process, gradually undertaken in this paper, delivers for each value of N (N=8..8192) the appropriate radix-2^r (r=8..512) and sub-radix-2^s (s=4..32) that lead to the architecture with the shortest critical path ($3\sqrt[3]{N/2}-3$) in adder stages. The couple (r,s) serves to partition the architecture so that maximum parallelism is exploited. As for area, our proposed architectures require as many hardware resources as modified Booth algorithm [13] with a critical path of N/2 [14][15][16][17]. For instance, a 64-bit two's complement finely pipelined multiplier requires a latency of seven clock cycles only (critical path composed of a series of 7 adders). FPGA implementation on Virtex-6 circuit of our 64-bit two's complement radix-2^{32} multiplier shows important gain ratios over Seidel [11] and Dimitrov [12] radix-2^8 algorithms. The respective gain ratios are enumerated as follows: 1.62, 1.71, 2.64 and 1.83, 1.71, 3.32 are obtained in terms of multiply-time, energy consumption per multiply-operation, and total gate count, respectively.

The paper is organized as follows. Section I outlines the main requirement specifications for a generalized radix-2^r multiplication. Section II introduces the new recursive multibit recoding multiplication algorithm, illustrated by two high-radix (2^8 and 2^{16}) recoding examples in Section III. Section IV introduces some preliminary steps toward an optimal partitioning of the multiplier architecture, while the optimal partitioning is presented in Section V. Section VI compares and discusses the implementation results. Finally, Section VII provides some concluding remarks and suggestions for future work.

II. The New Recursive Multibit Recoding Multiplication Algorithm

The equation (2.1.2) of the original multibit recoding algorithm presented in [2] does not offer hardware visibility. Let us rewrite it in a simpler hardware-friendly form, as follows:

$$Y=\sum_{j=0}^{\frac{N}{r}-1}\left(y_{rj-1}+2^0\,y_{rj}+2^1\,y_{rj+1}+2^2\,y_{rj+2}+\cdots+2^{r-2}\,y_{rj+r-2}-2^{r-1}\,y_{rj+r-1}\right)2^{rj}=\sum_{j=0}^{\frac{N}{r}-1}Q_j\,2^{rj}\quad(1)$$

Where $y_{-1}=0$ and $r\in \mathrm{N}^*$. For simplicity purposes and without loss of generality, we assume that r is a divider of N.

In equation (1), the two's complement representation of the multiplier Y is split into N/r two's complement slices (Q_j), each of r+1 bit length. Each pair of two contiguous slices has one overlapping bit. In literature, equation (1) is referred to by radix-2^r equation, to which corresponds a digit set $D(2^r)$ such as $Q_j\in D(2^r)=\{-2^{r-1},\ldots,0,\ldots,2^{r-1}\}$.

Thus, the signed multiplication between X and Y becomes: $X.Y=\sum_{j=0}^{\frac{N}{r}-1}X.Q_j.2^{rj}$ (2). Where each partial product can be expressed as follows: $X.Q_j.2^{rj}=(-1)^e.2^f.(m.X)$, with $m\in O(2^r)=\{1,3,\ldots,2^{r-1}-1\}$ such as $|O(2^r)|=2^{r-2}$. $O(2^r)$ represents the required set of odd-multiples of the multiplicand ($m.X$) for radix-2^r. Hence, the partial-product generation-process consists first in selecting one odd-multiple ($m.X$) among the whole set of pre-computed odd-multiples, which is then submitted to a hardwired shift of f positions, and finally conditionally complemented $(-1)^e$ depending on the bit sign e of Q_j term. Table I provides a picture on how the number of odd-multiples grows when the radix becomes higher. While lower $m.X$ can be obtained using just one addition ($3X$=$2X$+$1X$), the calculation of higher ones may require a number of computation steps ($11X$= $8X$+$2X$+$1X$).

TABLE I
Main Features of the Multibit Recoding Multiplication Algorith

Radix	Nbr. of Partial Products	Odd Multiples ($m.X$)
2^1	N	$1X$
2^2	$N/2$	$1X$
2^3	$N/3$	$1X$, $3X$
2^4	$N/4$	$1X$, $3X$, $5X$, $7X$
2^5	$N/5$	$1X$, $3X$, $5X$, $7X$, $9X$, $11X$, $13X$, $15X$

$|O(2^{r+1})|=2\times|O(2^r)|$. In radix-$2^r$, the multiplier Y is divided into N/r slices, each of r+1 bit length. Each pair of two contiguous slices has one overlapping bit.

To bypass the hard problem of odd-multiples, we exploit the fact that the N+1 bit-length *two's complement* multiplier Y on which equation (1) is applied, is composed of a series (N/r) of r+1 bit-length *two's complement* slices (Q_j digits) on which equation (1) can be recursively applied again. Based on this observation, let us announce the two following theorems accompanied with their respective proofs inserted in Appendix.

Theorem 1. *Any digit* $Q_j \in D(2^r)$ *can be represented in a combination of digits* $P_{ji} \in D(2^s)$, *such as s is a divider of r.*

When theorem (1) is applied to equation (1), it gives:

$$Y = \sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{s}-1} P_{ji}\, 2^{si}\right] 2^{rj} \quad (3) \quad ; \quad \text{where}$$

$P_{ji} \in D(2^s) = \{-2^{s-1}, \ldots, 0, \ldots, 2^{s-1}\}$ with

$O(2^s) = \{1, 3, \ldots, 2^{s-1}-1\}$ such as $\dfrac{|O(2^r)|}{|O(2^s)|} = 2^{ks}$ and

$$X.Y = \sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{s}-1} X.P_{ji}\, 2^{si}\right] 2^{rj} \quad (4)$$

Theorem 2. *Any digit* $Q_j \in D(2^r)$ *can be represented in a combination of digits* $P_{ji}+T_{jk}$ *such as* $P_{ji} \in D(2^s)$ *and* $T_{jk} \in D(2^t)$ *with* $s+t$ *a divider of* r, *and* $t < s$.

Likewise, when theorem (2) is applied to equation (1), we obtain: $Y = \sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{s+t}-1}\left[P_{ji} + T_{ji}\, 2^s\right] 2^{(s+t)i}\right] 2^{rj}$ (5). Where

$P_{ji} \in D(2^s) = \{-2^{s-1}, \ldots, 0, \ldots, 2^{s-1}\}$ with

$O(2^s) = \{1, 3, \ldots, 2^{s-1}-1\}$ and

$T_{ji} \in D(2^t) = \{-2^{t-1}, \ldots, 0, \ldots, 2^{t-1}\}$ with

$O(2^t) = \{1, 3, \ldots, 2^{t-1}-1\}$ such as $\dfrac{|O(2^r)|}{|O(2^{s+t})|} = 2^{k(s+t)}$

and $X.Y = \sum_{j=0}^{\frac{n}{r}-1}\left[\sum_{i=0}^{\frac{r}{s+t}-1}\left[X.P_{ji} + X.T_{ji}\, 2^s\right] 2^{(s+t)i}\right] 2^{rj}$ (6)

Theorem (1) and (2) allow an exponential reduction ($1/2^{ks}$ and $1/2^{k(s+t)}$, resp.) of the number of odd-multiples in equations (4) and (6) in comparison to equation (2), but at the expense of a linear increase (ks-1 and $k(s+t)$-1, resp.) in the number of additions. The advantage by far outweighs the cost, as practically shown in the next section.

The translation of equation (4) into architecture is depicted by Fig. 1.b, where each PPG_j (Q_j) is built up using r/s identical PPG_{ji} (P_{ji}). This is not the case for equation (6) which requires two different PPG_{ji} (P_{ji} and T_{ji}). Theorem (1) and (2) can be merged together to produce PPG_j made of a number of different PPG_{ji} (P_{ji}, T_{ji}, U_{ji}, V_{ji}, ...). This is the general case that is thoroughly studied in next sections in order to determine the optimal multiplier.

III. Two High Radix (2^8 and 2^{16}) Illustrative Examples

Theorems (1) and (2) permit to build up any high radix-2^r multiplication algorithm based on lower sub-radices, employing much less odd-multiples. The objective hereafter is to generate high radix-2^r multiplication *without* odd-multiples for a maximum reduction of multiplexer complexity inside PPG_j. To achieve such a goal, a number of odd-multiple free low-radix algorithms are used, such as Booth algorithm (radix-2^1) [18], modified Booth algorithm (radix-2^2) [13], Seidel et *al.* algorithms (radix-2^5 and radix-2^8) [11][19]. Booth and modified Booth recoding (McSorley algorithm [13]) can be derived from equation (3) for $(r,s)=(1,1)$ and $(r,s)=(2,2)$, respectively. They are respectively summarized as follows:

$$Y = \sum_{j=0}^{N-1}(y_{j-1} - y_j)2^j = \sum_{j=0}^{N-1} Q_j\, 2^j \quad (7)$$

With $D(2^1) = \{-1, 0, 1\}$ and $O(2^1) = \{1\}$

$$Y = \sum_{j=0}^{(N/2)-1}(y_{2j-1} + y_{2j} - 2y_{2j+1})2^{2j} = \sum_{j=0}^{(N/2)-1} Q_j\, 2^{2j} \quad (8)$$

With $D(2^2) = \{-2, -1, 0, 1, 2\}$ and $O(2^2) = \{1\}$

Seidel radix-2^5 recoding [11][19] is described as follows:

$Y = \sum_{j=0}^{(N/5)-1}\left[7.Q_j + P_j\right]2^{5j}$ (9) with $Q_j \in \{-2, -1, 0, 1, 2\}$;

$P_j \in \{-4, -2, -1, 0, 1, 2, 4\}$ and $O(2^5) = \{1\}$.

And Seidel radix-2^8 recoding is given by the following equation: $Y = \sum_{j=0}^{(N/8)-1}\left[11^2.Q_j + 11.P_j + T_j\right]2^{8j}$ (10) with

$Q_j \in \{-2, -1, 0, 1, 2\}$; $P_j, T_j \in \{-16, -8, -4, -2, -1, 0, 1, 2, 4, 8, 16\}$ and $O(2^8) = \{1\}$. Note that while equations (9) and (10) are odd-multiple free since all included digits are power of 2, they require a post-accumulation to deal with odd numbers (7, 11 and 121). Thus, a number of extra-adders are needed.

Optimized higher radices are obtained as follows.

A. Our new radix-2^8 recoding

Based on theorem (2), each 8+1 bit slice is split into 5+1, 2+1, and 1+1 overlapping slices using Seidel radix-2^5, McSorley radix-2^2, and Booth radix-2^1 algorithms, respectively. The new recoding is given by the following equation: $Y = \sum_{j=0}^{(N/8)-1}\left[\left(7.Q_j + P_j\right) + \left(R_j + S_j\, 2^2\right).2^5\right].2^{8j}$ (11)

With $Q_j \in \{-2, -1, 0, 1, 2\}$; $P_j \in \{-4, -2, -1, 0, 1, 2, 4\}$;

$R_j \in \{-2, -1, 0, 1, 2\}$; $S_j \in \{-1, 0, 1\}$ and $O(2^8) = \{1\}$

B. Our new radix-2^{16} recoding

Likewise, using theorem (2), each 16+1 bit slice is split into 8+1, 5+1, 2+1, and 1+1 overlapping slices using Seidel radix-2^8 and radix-2^5, McSorley radix-2^2, and Booth radix-2^1 algorithms, respectively. The new recoding is described by the following equation:

$$Y = \sum_{j=0}^{\frac{N}{16}-1}\left[\left(11^2.Q_j + 11.P_j + T_j\right) + \left(7.R_j + S_j\right).2^8 + \left(U_j + V_j\, 2^2\right).2^{13}\right].2^{16j} \quad (12)$$

with $Q_j \in \{-2, -1, 0, 1, 2\}$;

$P_j, T_j \in \{-16, -8, -4, -2, -1, 0, 1, 2, 4, 8, 16\}$;

$R_j \in \{-2, -1, 0, 1, 2\}$; $S_j \in \{-4, -2, -1, 0, 1, 2, 4\}$;

$U_j \in \{-2, -1, 0, 1, 2\}$; $V_j = \{-1, 0, 1\}$ and $O(2^{16}) = \{1\}$

In our preceding work [20], we pursued this combination process farther and generated a series of higher radix (2^{24}, 2^{32}, ...) recoding schemes with $O(2^r) = \{1\}$. However, what still remains unknown is to determine, for a given N value, the proper radix (2^r) that leads to the optimal architecture.

The translation of equations (11) and (12) into architectures is depicted in Fig. 2.a and 2.b, respectively.

All Dimitrov algorithms developed in [12] are unsigned. For an equitable comparison, we had to develop a new two's complement radix-2^8 recoding version with $O(2^8)=\{1,3,5,7\}$ based on Dimitrov unsigned radix-2^7 recoding (mult_7b2d in [12]) with $O(2^7)=\{1,3,5,7\}$. The new recoding is:

$$Y=\sum_{j=0}^{(n/8)-1}\left(2^k\cdot Q_j+(-1)^e\,2^h\cdot P_j\right)(-1)^{8j+7}\,2^{8i} \qquad (13)$$

With $Q_j, P_j \in \{1,3,5,7\}; k,h \in \{0,1,2,3,4,5,6,7\}$ *and* $e \in \{0,1\}$

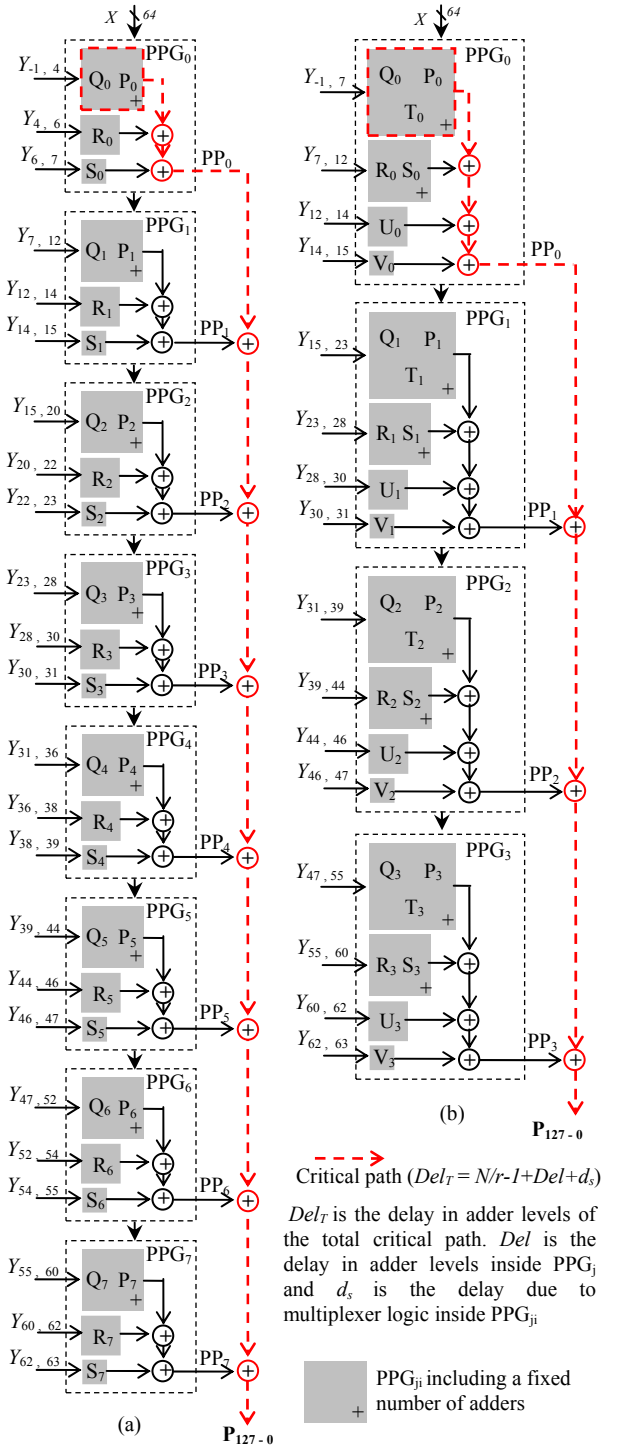

Fig. 2. Two's complement 64×64 bit multiplier.
(a) Radix-2^8 multiplier. Space partitioning according to equation (11)
(b) Radix-2^{16} multiplier. Space partitioning according to equation (12)

For the comparative study, our proposed algorithms (eq. 11 and 12) as well as Seidel and Dimitrov algorithms (eq. 10 and 13, resp.) are first analytically characterized and then physically implemented.

C. *Analytical characterization of area and speed*

Prior implementation, we need to develop a generalized theoretical model which predicts area and speed features of each recoding algorithm with respect to *N* and *r* values.

1) *Area*

Three basic components are necessary for the implementation of RTL multipliers:

- multiplexers (*Mux*1) to recode the digit terms (Q_j,P_j,...) included in the recoding expression;
- shifters (*Mux*2) for partial product generation;
- and adders for partial product summation.

Whereas the exact number of adders can be known in advance, we need to develop heuristics for the two others. The total multiplexer complexity (*Mux*1) of a radix-2^r multiplier depends on:

- the number (N/r) of PPG_j;
- the number (i) of lower sub-radices (2^1, 2^2, 2^5, and 2^8) used to build up the higher radix-2^r. To each sub-radix-2^s used (PPG_{ji}) corresponds an RTL "case statement" that recodes the digit terms (Q_{ji},P_{ji},T_{ji},…) present in the equation;
- the number of entries (e_s+1) in each "case statement" corresponding to each sub-radix-2^s;
- the number (d_s) of digit terms (Q_{ji},P_{ji},T_{ji},…) that figures in each "case statement";
- and on the number of necessary odd-multiples ($|O_s|$) used to calculate the digit terms.

Hence, we can announce that: $Mux1=\frac{N}{r}\cdot\sum_i\left(2^{e_s+1}\cdot d_s\cdot|O_s|\right)$

For Dimitrov algorithm (eq. 13), this gives: r=8, i=1, e_s=8, d_s=2, and $|O_s|$=4. Thus, *Mux*1 = 512 *N*.

The synthesis of the RTL "shift statement" infers multiplexers whose complexity depends on the number (p_{sj}) of different shift positions for all odd-multiples involved in the calculation of each digit term (j). Thus, we can write: $Mux2=\frac{N}{r}\cdot\sum_i\sum_j\left(p_{sj}\cdot|O_{sj}|\right)$. For Dimitrov algorithm (eq. 13), this gives: r = 8, i=1, j=2, $p_{s1}=p_{s2}$ =8, and $|O_{s1}|$ = $|O_{s2}|$ = 4. Thus, *Mux*2=8*N*. Hence, the total multiplexer complexity becomes: Mux_T = *Mux*1+*Mux*2=520*N*.

A *N*-bit radix-2^r multiplier generates N/r PP. Thus, The total number of adders comprises:

- $(N/r)-1$ adders to sum the N/r PP;
- plus the necessary adders inside each PPG_j to accumulate the intermediate PP issuing from PPG_{ji};
- plus a number of adders included inside each PPG_{ji} depending on the recoding scheme used.

For example, in Seidel algorithm (eq. 10), the term $11^2Q_{ji}+11P_{ji}+T_{ji}$ is calculated as follows: $\left(2^7Q_{ji}-2^3Q_{ji}+Q_{ji}\right)+\left(2^3P_{ji}+2^2P_{ji}-P_{ji}\right)+T_{ji}$, which requires 6 adders for post-accumulation operation [11][19]. Hence, the total number of necessary adders is: $Add_T=(N/8)-1+6(N/8)=(7/8)N-1$.

TABLE II
MAIN FEATURE COMPARISON

Features	Our recoding algorithms		McSorley [13] Eq. (8)	Seidel [11] [19] Eq. (10)	Dimitrov [12] Eq. (13)
	Eq. (11)	Eq. (12)			
Radix	2^8	2^{16}	2^2	2^8	2^8
Del_T	$\frac{N}{8}+3+d_5$	$\frac{N}{16}+8+d_8$	$\frac{N}{2}-1+d_2$	$\frac{N}{8}+5+d_8$	$\frac{N}{8}+d'_8$
Mux_T	$19N$	$106N$	$5N$	$194N$	$520N$
Add_T	$\frac{5N}{8}-1$	$\frac{6N}{8}-1$	$\frac{N}{2}-1$	$\frac{7N}{8}-1$	$\frac{N}{4}-1$

N is the operand size and 2^r is the radix used. Del_T is the total delay in terms of adder levels in the critical path of a linear reduction tree. d_s is the delay due to multiplexer logic inside PPG_{ji}. d_s depends on *Mux* factor ($d_1<d_2<d_5<d_8<d'_8$). $Mux_T=(N/r)Mux$, where *Mux* is an estimation of the multiplexer logic required by PPG_j. Add_T is the total number of adders required in the whole multiplier.

2) Delay

The total delay (Del_T) along the critical path is the summation of PPG_j delay and reduction tree delay. Based on the total number of adders (Add_T), the critical path of the multiplier in terms of logic levels is: $Del_T= N/r-1+Del+d_s$, where *Del* is the delay due to adder stages inside PPG_j and d_s is the delay due to multiplexer logic inside PPG_{ji}. This latter depends on *Mux* factor of used PPG_{ji} (2^1, 2^2, 2^5, or 2^8). Therefore, $d_1 < d_2 < d_5 < d_8$. Note that d_s is fixed and *Del* depends on r and s values. For instance, according to equation (10), Seidel algorithm exhibits a critical path of: $Del_T= N/8-1+6+d_8=N/8+5+d_8$. Table II provides the area occupation and delay for each recoding algorithm.

D. Physical implementation

All recoding schemes mentioned in Table II underwent several verification steps. First all equations were validated with a random C-program. Then, they were implemented at RTL level in Verilog-2001 (IEEE 1364) as technology-independent reusable IP-cores [1], using *exactly* the same optimized coding style for an equitable comparison. They are compile-time reconfigurable according to N and r. Reader is referred to [11], [19], and [12] for recoding tables used in equations (9), (10), and (13), respectively.

All RTL codes went through a severe cycle-accurate functional verification procedure using Modelsim SE-6.3f logic simulator. They were first challenged against a set of special and severe test cases, and then submitted to a random test for a very large number of vectors. After a successful functional verification, physical tests were performed. They were integrated into an FPGA evaluation board for an ultimate validation. Afterwards, all equations were synthesized and mapped to the same Virtex-6 FPGA circuit (xc6vsx475t-2ff1156) using Xilinx ISE 13.2 release version [21]. We used for comparison a two's complement 64×64 bit parallel multiplier. The implementation results are grouped in Table III.

TABLE III
IMPLEMENTATION RESULTS OF A TWO'S COMPLEMENT 64-BIT PARALLEL MULTIPLIER ON XILINX XC6VSX475T-2FF1156 CIRCUIT

Results	Our recoding algorithms		McSorley [13] Eq. (8)	Seidel [11] [19] Eq. (10)	Dimitrov [12] Eq. (13)
	Eq. (11)	Eq. (12)			
Area[1]	3219	4659	2103	5251	6599
Energy[2]	1.63	2.11	1.46	2.49	2.48
Speed[3]	52.4	49.34	30.04	48.62	43.17

Synthesis tool was forced to map RTL code to distributed slices of FPGA and avoid mapping to builtin 18x18 bit hardwired multipliers (DSP slices).
1: Area occupation in number of Virtex-6 slices. 2: Energy consumption per multiplication operation (pJ). 3: Million multiplications per second (MMPS).

Although Dimitrov recoding exhibits the shortest critical path in adder stages ($N/8$), the impact of multiplexer logic (d'_8) on the total performance is important (Table III). Besides, it is the most area consumer despite the fact that it employs the lowest number of adders ($N/4$-1). Adversely, Seidel algorithm is the most adder consumer ($7N/8$-1). To determine which factor, Mux_T or Add_T, exerts more influence on area occupation, let us compare their respective ratios for Seidel and Dimitrov algorithms: $Mux_T(Eq.13)/Mux_T(Eq.10)=2.7$ and $Add_T(Eq.10)/Add_T(Eq.13)=3.5$.

Significant conclusion: the area occupation is dominated by Mux_T factor, and becomes larger as Mux_T number becomes higher (Table II and III). This correlation is advantageously used to minimize area occupation as will be shown in the next section.

McSorley algorithm (eq. 8) is the least area consumer and the slowest recoding scheme for any value of N. The best area/speed compromise for N=64 is given by our recoding scheme based on equation (11). However, this latter will be outperformed by equation (12) for larger values of N (N>64) since a higher radix (2^{16}) is employed.

While energy consumption is function of the switched capacitance, Table III shows a direct correlation between area occupation and energy consumption. Making Mux_T indicator lower, will result in a less energy-consumer recoding algorithm.

Finally, based on theory and implementation results, we conclude that the best tradeoff related to our recoding schemes depends on N and r values. For larger N values (N>64), larger radices are necessary to reduce the critical path. But for larger radices (r>16) we need to duplicate some of the elementary PPG_{ji} (2^1,2^2,2^5,2^8) to build up the radix-2^r PPG_j. Therefore, at this level a relevant question arises: given N, what is the value of r and its corresponding elementary PPG_{ji} configuration (optimal partitioning of PPG_j) that leads to the shortest critical path (Del_{Tmin}) with minimum hardware resources (Mux_{Tmin})? The answer to this question is given in the next sections.

IV. PRELIMINARY STUDY TO AN OPTIMAL PARTITIONNING

We extend the recoding-space of our equations (11) and (12) to the general case as follows: each $r+1$ bit slice is recoded using a, b, c, d instances of radix 2^8, 2^5, 2^2, 2^1 algorithms, respectively, such that $8a+5b+2c+d=r$. To this recoding scheme corresponds the following equation:

$$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{a-1}A_i^j.2^{8i}+\sum_{i=0}^{b-1}B_i^j.2^{5i+8a}+\sum_{i=0}^{c-1}C_i^j.2^{2i+8a+5b}+\sum_{i=0}^{d-1}D_i^j.2^{i+8a+5b+2c}\right].2^{rj} \quad (14)$$ where

$A_i^j=11^2.Q_i^j+11.P_i^j+T_i^j$ with $Q_i^j\in\{-2,-1,0,1,2\}$ and $P_i^j, T_i^j\in\{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$;

$B_i^j=7.R_i^j+S_i^j$ with $R_i^j\in\{-2,-1,0,1,2\}$ and $S_i^j\in\{-4,-2,-1,0,1,2,4\}$;

$C_i^j=y_{2i-1}^j+y_{2i}^j-2y_{2i+1}^j$ with $C_i^j\in\{-2,-1,0,1,2\}$;

finally $D_i^j=y_{i-1}^j-y_i^j$ with $D_i^j\in\{-1,0,1\}$.

The translation of equation (14) into architecture is depicted in Fig. 1.b (*top view only*), where each PPG_j is built

up using a mixture of four different PPG_{ji} depending on the quadruplet (*a,b,c,d*) as illustrated by Fig. 3. For instance, to equations (11) and (12) correspond (0,1,1,1) and (1,1,1,1), respectively. Note that because of the general nature of equation (14), the d_s term of Del_T is equal to max(d_8,d_5,d_2,d_1) of used PPG_{ji}.

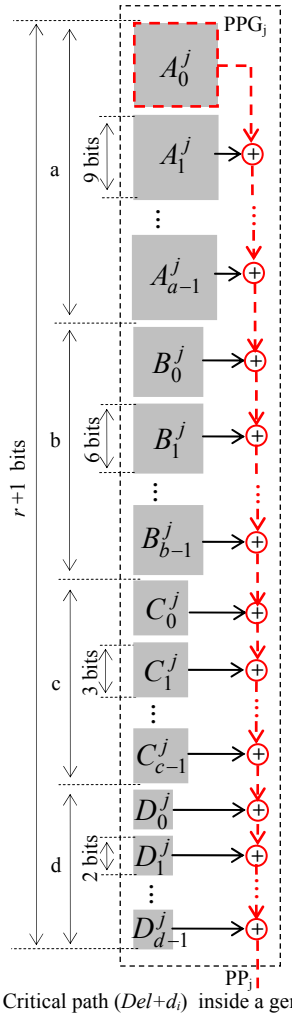

Fig. 3. Critical path ($Del+d_i$) inside a generalized PPG_j

Given *N* and *r*, to determine the optimal partitioning of the whole multiplier (global optimum since PPG_j are identical), we need to find first the quadruplet (*a,b,c,d*) that satisfies the condition $8a+5b+2c+d=r$ and leads to the PPG_j with minimum hardware ressources (Mux_{min}) and the shortest critical path (Del_{min}). As it is not sure that such a solution exists, we are using composite metrics A^iT^j of area (A) and delay (T) for i and j varying from 0 to 5 [22]. A total of 11 metrics (A, A^5T, A^4T, A^3T, A^2T, AT, AT^2, AT^3, AT^4, AT^5, T) are used. The A metric alone delivers the best area solution (Mux_{min}), while T metric provides the best delay solution (Del_{min}). In between(A^iT^j), more-or-less balanced solutions are obtained. The implementation of this solution requires the (*Mux*, *Del*) couple (Table IV) corresponding to each basic recoding algorithm (2^8,2^5,2^2,2^1). Because of an explosive number of possible combinations (*N*>>), the solution space is *exhaustively* explored using a deterministic C-program for *r* varying from 8 to 1024. The obtained results are reported in Table V.

TABLE IV
DELAY AND MULTIPLEXER COMPLEXITY OF BASIC RADICES: STEP #1

Algorithm	*Del*	*Mux*
2^1	0	5
2^2	0	10
2^5	2	133
2^8	6	1548

Mux values are extracted from the heuristic developed in Section III. Ex: 1548=194 × 8.

TABLE V
OPTIMAL PPG_j SOLUTION (a,b,c,d) LEADING TO THE OPTIMAL RADIX-2^r MULTIPLIER ACCORDING TO COMPOSITE METRICS A^iT^j

r size (bits)	Criteria	Instance Number *a*	*b*	*c*	*d*	*Del*	*Mux*	Del_{min}	Mux_{min}
8	A – T	0	0	4	0	3	40	3	40
16	$A – AT^5$	0	0	8	0	7	80	5	80
	T	0	3	0	1	5	404		
32	$A – AT^3$	0	0	16	0	15	160	8	160
	$AT^4 – T$	0	6	1	0	8	808		
64	$A – AT^2$	0	0	32	0	31	320		
	$AT^3 – AT^5$	0	12	2	0	15	1616	13	320
	T	8	0	0	0	13	12384		
128	$A – AT^2$	0	0	64	0	63	640		
	$AT^3 – AT^5$	0	25	1	1	28	3340	21	640
	T	16	0	0	0	21	24768		
256	A – AT	0	0	128	0	127	1280		
	$AT^2 – AT^5$	0	51	0	1	53	6788	37	1280
	T	32	0	0	0	37	49536		
512	A – AT	0	0	256	0	257	2560		
	$AT^2 – AT^5$	0	102	1	0	104	13576	69	2560
	T	64	0	0	0	69	99072		
1024	A – AT	0	0	512	0	512	5120		
	$AT^2 – AT^5$	0	204	2	0	207	27152	133	5120
	T	128	0	0	0	133	198144		

A – T: all the metric span A, A^5T, A^4T, A^3T, A^2T, AT, AT^2, AT^3, AT^4, AT^5, T. To A and T metrics correspond respectively the minimal values Mux_{min} and Del_{min} that serve as reference for the optimization process.

As conclusion, optimal area solutions ($Mux=Mux_{min}$) are exclusively based on radix-2^2 algorithm (*0,0,c,0*), but they are excessively slow ($Del>>Del_{min}$). While optimal speed solutions ($Del=Del_{min}$) are entirely composed of radix-2^8 algorithm (*a,0,0,0*), but they are exaggeratedly large ($Mux>>Mux_{min}$). Finally, balanced area/speed solutions are mainly based on radix-2^5 algorithm with at most one or two instances of radices 2^1 and 2^2 algorithms (*0,b,c,d*). However, even the "balanced" solution is not really balanced enough since the mean values of *Del* and *Mux* are $1.4\times Del_{min}$ and $5.2\times Mux_{min}$, respectively. The reason is due to the large disparity between *Mux* values of the basic radices (Table IV). To correct this disequilibrium, we replace respectively the two Seidel radix-2^8 and 2^5 expressions (A_i^j and B_i^j) included in equation (14) by their *mathematically equivalent* counterparts as follows: $A_i^j = \sum_{k=0}^{3} 2^{2k} C_k^{ji}$ and $B_i^j = D_0^{ji} + 2C_0^{ji} + 2^3 C_1^{ji}$. These new expressions are radix-2^8 and 2^5, respectively. They produce respectively the same intermediary partial products at PPG_{ji} output as their Seidel counterparts. In fact A_i^j is formed by a succession of four instances of McSorley algorithm, while B_i^j is composed of one instance of Booth algorithm followed by two instances of McSorley algorithm. *Del* and *Mux* values of the new basic radices are grouped in Table VI. Results delivered by the deterministic C-program are reported in Table VII. All solutions are optimal since $Del=Del_{min}$ and $Mux=Mux_{min}$. They are all based on radix-2^8 algorithm (*a,0,0,0*). In case *r* is not a multiple of 8, optimal solutions are also obtained, composed mainly of radix-2^8 algorithm with at most one instance of radix-2^1, 2^2 or 2^5 algorithms, depending on the remainder of *r* by 8 division.

TABLE VI
DELAY AND MULTIPLEXER COMPLEXITY OF THE NEW BASIC RADICES: STEP #2

Algorithm	*Del*	*Mux*
2^1	0	5
2^2	0	10
2^5	2	25
2^8	3	40

TABLE VII

OPTIMAL PPG$_J$ SOLUTION (a,b,c,d) LEADING TO THE OPTIMAL RADIX-2^r MULTIPLIER ACCORDING TO COMPOSITE METRICS A^IT^J

r size (bits)	Instance Number				Del	Mux	Del_{min}	Mux_{min}
	a	b	c	d				
8	1	0	0	0	3	40	3	40
16	2	0	0	0	4	80	4	80
32	4	0	0	0	6	160	6	160
64	8	0	0	0	10	320	10	320
128	16	0	0	0	18	640	18	640
256	32	0	0	0	34	1280	34	1280
512	64	0	0	0	66	2560	66	2560
1024	128	0	0	0	130	5120	130	5120

The new results are so interesting that we are encouraged to pursue further the optimization process using higher basic sub-radices (s>8) to reduce the total delay (Del_T) of the multiplier. Let us this time replace A_i^j and B_i^j as follows: $A_i^j = \sum_{k=0}^{7} 2^{2k} C_k^{ji}$ and $B_i^j = \sum_{k=0}^{3} 2^{2k} C_k^{ji}$. We eliminate radix-$2^5$ since it can be derived from radix-2^1 and 2^2. The new Del and Mux values of basic radices are grouped in Table VIII.

TABLE VIII

DELAY AND MULTIPLEXER COMPLEXITY OF THE NEW BASIC RADICES: STEP #3

Algorithm	Del	Mux
2^1	0	5
2^2	0	10
2^8	3	40
2^{16}	7	80

The C-program shows up even more interesting results since starting from r≥64 (Table IX), lower delays are obtained with the same multiplexer complexities as the ones reported in Table VII. Based on the obtained results, we pushed farther the optimization process using even higher basic sub-radices (s=16..32).

All optimal solutions come either on the form $(a,0,0,0)$ or $(0,b,0,0)$. At this level we can draw a *significant conclusion*: since the optimal solution is always in the form $(a,0,0,0)$ or $(0,b,0,0)$ with $a=2k$ and $b=2k'$, there exists an integer $s=2k''$ such as either $(s,0,0,0)$ or $(0,s,0,0)$ is the optimal solution.

Consequently, equation (14) is rewritten accordingly, as follows:

$$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s}-1} \left[\sum_{k=0}^{\frac{s}{2}-1} C_k^{ji} . 2^{2k} \right] 2^{si} \right] 2^{rj} \qquad (15)$$

and $C_k^{ji} = y_{2k-1}^{ji} + y_{2k}^{ji} - 2y_{2k+1}^{ji}$ with $C_k^{ji} \in \{-2,-1,0,1,2\}$.

Based on heuristic developed in Section III, multiplexer complexity of equation (15) for the whole multiplier is always equal to Mux_T=10×$N/2$=5N for any value of r and s. As for the multiplier delay (Del_T), we need to determine the couple (r,s) that leads to the shortest critical path in terms of adder levels. This is what is achieved in the next section.

TABLE IX

OPTIMAL PPG$_J$ SOLUTION (a,b,c,d) LEADING TO THE OPTIMAL RADIX-2^r MULTIPLIER ACCORDING TO COMPOSITE METRICS A^IT^J

r size (bits)	Instance Number				Del	Mux	Del_{min}	Mux_{min}
	a	b	c	d				
8	0	1	0	0	3	40	3	40
16	0	2	0	0	4	80	4	80
32	0	4	0	0	6	160	6	160
64	4	0	0	0	10	320	10	320
128	8	0	0	0	14	640	14	640
256	16	0	0	0	22	1280	22	1280
512	32	0	0	0	38	2560	38	2560
1024	64	0	0	0	70	5120	70	5120

→ : Optimal solution moved from $(0,b,0,0)$ to $(a,0,0,0)$

V. THE OPTIMAL PARTITIONNING

The total delay (Del_T) of the whole multiplier related to equation (15) is: Del_T= N/r-1+Del+d_2 where Del is the PPG$_j$ delay equal to $(r/s-1)+(s/2-1)$, and d_2 is the multiplexer delay corresponding to the recoding logic of radix-2^2. Thus, Del_T= $N/r+r/s+s/2-3+d_2$.

The optimal delay with regard to r is obtained for (r,s) couples satisfying $\partial(Del_T)/\partial r = 0$, which gives $r = \sqrt{s.N}$. When r is substituted by $\sqrt{s.N}$ into Del_T expression, we obtain: $Del_T = 2\sqrt{N/s} + s/2 - 3 + d_2$. Likewise, the optimal delay with regard to s is obtained for s value satisfying $\partial(Del_T)/\partial s = 0$. We obtain $s = 2\sqrt[3]{N/2}$. Hence, the optimal delay becomes: $Del_T = 3\sqrt[3]{N/2} - 3 + d_2$.

Finally, we conclude that the optimal N-bit multiplier, in comparison to equation (8) [13], relies on the new triple recursive equation (15) with (r,s)=($\sqrt[3]{2.N^2}$, $2\sqrt[3]{N/2}$).

Table X provides the s and r values that lead to the optimal partitioning with respect to the operand size N. The values s and r correspond to the number of multiplier bits that are treated simultaneously inside each PPG$_{ji}$ and each PPG$_j$, respectively. For N=64, the optimal partitioning is obtained with (r,s)=(32,8) as illustrated by Fig. 4. Whereas equations (15) and (8) require the same amount of hardware resources (Mux_T , Add_T)=(320,31), they exhibit different critical paths: 7 and 31 in terms of adder levels, respectively.

VI. DISCUSSION OF THE IMPLEMENTATION RESULTS

We proved via FPGA implementation (Table III) how much accurate are the area heuristics developed in Section III (Table II). Based on this, we have undertaken a gradual theoretical optimization process that yielded to equation (15). This latter is implemented on FPGA with N=64, and the results in terms of multiply-time, energy consumption per multiply-operation, and total gate count, are as follows: 78.98 MMPS, 1.45pJ and 1987 slices, respectively. Compared to implementation results of Seidel and Dimitrov algorithms (Table III), gain ratios of 1.62, 1.71, 2.64 and 1.83, 1.71, 3.32 are obtained, respectively. A 64-bit multiplier generated by Xilinx Coregen exhibits 75.86 MMPS and consumes twelve 18×18 bit DSP-slice multipliers.

The real reasons behind these important results are cleared up as follows.

TABLE X

THE OPTIMAL PARTITIONING VERSUS OPERAND SIZE N

N (bits)	Our recoding Equation (15)			McSorley[13] Equation (8)	Seidel[11][19] Equation (10)	Dimitrov [12] Equation (13)
	s	r	Del_T	Del_T	Del_T	Del_T
8	4	8	2	3	6	1
16	4	8	3	7	7	2
32	8	16	5	15	9	4
64	8	32	7	31	13	8
128	8	32	9	63	21	16
256	16	64	13	127	37	32
512	16	128	17	255	69	64
1024	16	128	21	511	133	128
2048	32	256	28	1023	261	256
4096	32	512	35	2047	517	512
8192	32	512	45	4095	1029	1024

s value corresponds to the number of bits that are treated simultaneously inside each PPG$_{ji}$, while r value indicates the number of bits that are processed simultaneously inside each PPG$_j$. d_s is not included in Del$_T$ since d_2<d_8<d'_8.

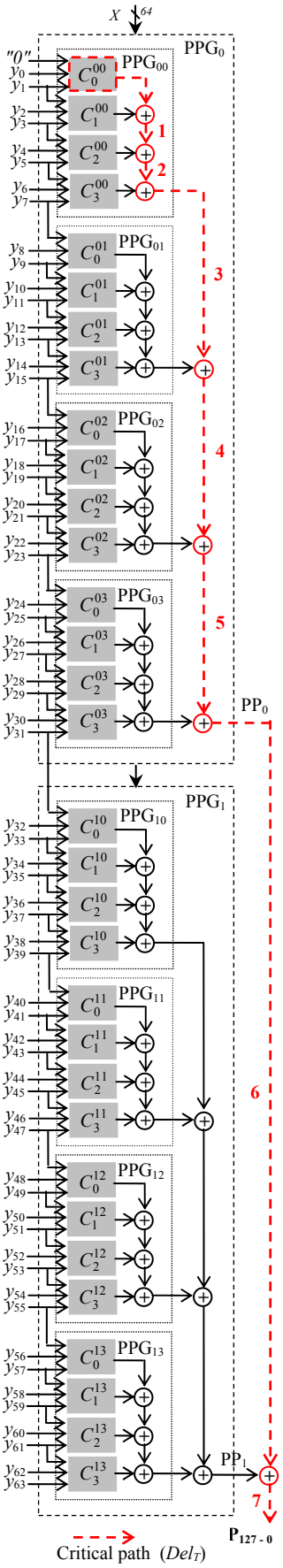

Fig. 4. Optimal partitioning of a two's complement 64×64 bit radix-2^{32} parallel multiplier based on equation (15) with $(r,s)=(32,8)$.

A. Area occupation

For operand size N=64, equation (15) is a composite radix-2^{32} algorithm (Table X), where each PPG_j processes simultaneously 32+1 inputs that are split on four sub-radix-2^8 PPG_{ji} made of four instances (C_k^{ji}) of McSorley algorithm (Fig. 4). Seidel and Dimitrov algorithms are rather radix-2^8 algorithms, based on mono-bloc PPG_j.

In fact, although radix-2^8 PPG_{ji} of equation (15) and radix-2^8 PPG_j of Seidel and Dimitrov are based on different recoding schemes, they are *mathematically equivalent* since they produce the same partial product PP_{ji}/PP_j. Based on theory (Table II) and implementation results (Table III), Dimitrov recoding is the most space consuming due to the use of odd-multiples of the multiplicand. On the other hand, Seidel recoding does not require odd-multiples, but since 9 inputs are treated simultaneously in a mono-bloc PPG_j, a large amount of multiplexer resources is needed to recode the 2^9=512 input combinations. Finally, radix-2^8 PPG_{ji} of equation (15) is the least area consumer because it does not employ odd-multiples and requires a small amount of multiplexers as the total number of input combinations in each radix-2^8 PPG_{ji} is equal to 8+8+8+8=32. Note that the three recoding schemes are incorporating a number of adders in their PPG_{ji}/PPG_j which is 3, 6, and 1 for equation (15), Seidel and Dimitrov algorithms, respectively.

Significant conclusion: the area occupation is dominated by the *Mux* factor, and becomes larger as *Mux* number becomes higher.

B. Delay

Using higher radices (r>>) will certainly shortens the critical path. However, for high r values, mono-bloc PPG_j recoding induces an important delay (d_s) due to the high density of multiplexer logic that significantly degrades the whole performance of the multiplier. This is clearly illustrated by Dimitrov radix-2^8 recoding whose critical-path totalizes 8 adder levels but exhibits a lower multiply rate (43.17 MMPS) compared to Seidel recoding that have a critical-path composed of 13 adder levels but shows a more interesting rate (48.62 MMPS) due to lower multiplexer complexity (Table II and III). As for equation (15), since a composite PPG_j is used, d_s is equal to d_2 (C_k^{ji} delay) which is the smallest delay ($d_2 < d_5 < d_8$). Besides, the critical path goes through the smallest number (7) of adder stages, exploiting maximum parallelism that can be provided by the triple-recursive equation (15). Thus, it is not surprising that equation (15) achieves the best performance (78.98 MHz), even when compared to Xilinx Coregen multiplier based on DSP-slices (75.86 MHz). A double-recursive (s=2) version of equation (15) served to design a scalable 16-bit setpoint Finite-Word-Length PID controller, employing five multiplication cores. The implementation results outperformed the published ones at all levels [23].

Significant conclusion: using composite recoding in conjunction with an optimal partitioning (r and s values) provides the shortest critical path.

Equation (15) shows high aptitude for pipelining. Two finely and coarsely grained systolic architectures for 64-bit multiplier are depicted in Fig. 5.a and Fig. 5.b, respectively. Fig. 5.a architecture is more suitable for high throughput applications, with 7 clock-cycle latency.

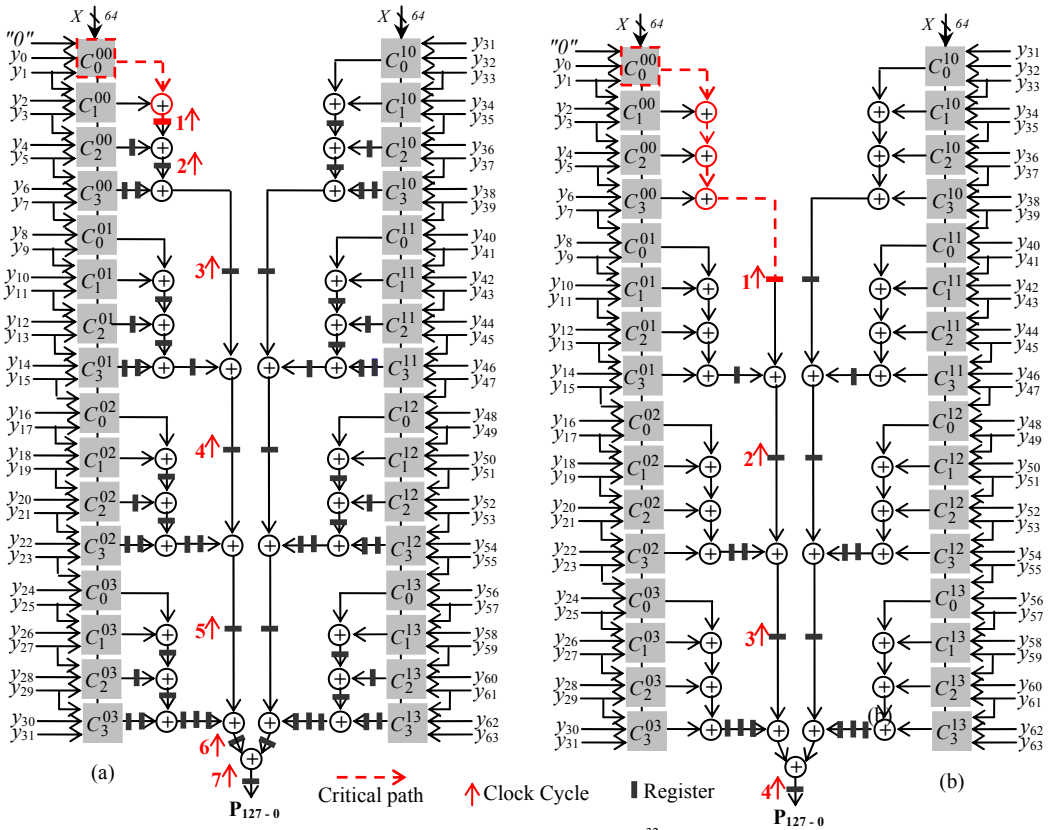

Fig. 5. Space/Time partitioning of a two's complement 64×64 bit radix-2^{32} parallel multiplier based on equation (15).
(a) High-throughput finely-grained systolic architecture; (b) Low-latency coarsely-grained systolic architecture.

VII. Conclusion and Futur Work

Upon the basis of the new multibit recoding multiplication algorithm, we developed optimal parallel multipliers with shortest critical paths and minimum hardware resources for any value of operand size *N*. We demonstrated by theory and FPGA implementation the superiority of our high-radix algorithms over their existing counterparts. Because exploiting the maximum parallelism inherent in multiply operation, our look-up-table based multiplier (eq. 15) is even speed-competitive with Xilinx's hardwired multiplier employing DSP-Slices (18×18 bit full-custom multipliers).

More importantly, we demonstrated also that the current trend relying upon minimal number-bases for the development of high radix-2^r recoding ($r \geq 8$) with mono-bloc PPG requires an excessive amount of multiplexer resources, which offsets speed and power benefits of the compressor factor N/r. On the other hand, we proved that composite PPG based on the new recursive multibit recoding algorithm is the best realistic alternative.

The topology of our proposed recoding schemes shows high capabilities for pipelining which can be finely or coarsely grained to satisfy both high throughput and low latency applications. A radix-2^{32} 64-bit parallel multiplier was finely pipelined, resulting in a systolic architecture with seven clock-cycle latency.

While the theoretical concept was validated using FPGA as a preliminary step, an ASIC implementation based on a standard-cell library is necessary for an ultimate validation of the whole optimization work. This issue will be explored in the near future, and we intend to report our results in a forthcoming paper.

Acknowledgment

This work is supported by "Centre de Développement des Technologies Avancées, CDTA," Algiers, Algeria, under project contract number: 21/CRSOC/DMN/CDTA/2011. The project progresses under a close cooperation with "Franche Comté Electronique Mécanique Thermique et Optique-Sciences et Technologies, FEMTO-ST " Besançon, France.

The authors wish to thank T. Hilaire and B. Djezzar for their careful review of this manuscript.

References

[1] Reports on System Drivers of the International Technology Roadmap for Semiconductors (ITRS), 2009 and 2010. Available: www.itrs.net/reports.html

[2] H. Sam, and A. Gupta, "A Generalized Multibit Recoding of Two's Complement Binary Numbers and its Proof with Application in Multiplier Implementation," IEEE Trans. on Computers, vol. 39, N° 8, August 1990.

[3] G. Kim et *al.*, "A Low-Energy Hybrid Radix-4/-8 Multiplier for Portable Multimedia Applications," Proceedings of IEEE International Symposium on Circuits and Systems, (ISCAS), pp. 1171-1174, Rio de Janeiro, Brazil, May 15-18, 2011.

[4] B.J. Benschneider et al, "A Pipelined 50MHz CMOS 64-Bit Floating-Point Arithmetic Processor," IEEE Journal of Solid-State Circuits, vol. (24) 5, pp. 1317-1323, October 1989.

[5] C.F. Webb et al, "A 400-MHz s/390 Microprocessor," IEEE Journal of Solid-State Circuits, vol. (32) 11, pp. 1665-1675, November 1997.

[6] J. Clouser et al, "A 600-MHz Superscalar Floating-point Processor," IEEE Journal of Solid-State Circuits, vol. (34) 7, pp. 1026-1029,July 1999.

[7] R. Senthinathan et al, "A 650-MHz, IA-32 Microprocessor with Enhanced Data Streaming for Graphics and Video," IEEE Journal of Solid-State Circuits, vol. (34) 11, pp. 1454-1465, November 1999.

[8] A. Scherer et al, "An Out-of-Order Tree-Way Superscalar Multimedia Floating Point Unit," Proceeding of IEEE International Solid-State Circuits Conference (ISSCC), pp. 94-95, 1999.

[9] Intel Corp., "Intel 64 and IA-32 Architectures Software Developers Manual," volume 1, order number 253668, Copyright May 2011.

[10] R.J. Rieldlinger, "A 32 nm 3.1 Billion Transistor 12-Wide-Issue Itanium Processor for Mission-Critical Servers," Proceedings of IEEE International Solid-State Circuits Conference (ISSCC), pp. 84-86, San Francisco, CA ,USA, February 20-24, 2011.

[11] P.M. Seidel, L. D. McFearin, and D.W. Matula, "Secondary Radix Recodings for Higher Radix Multipliers," IEEE Trans. on Computers, vol. 54, N°2, February 2005.

[12] V.S. Dimitrov, K.U. Järvinen, and J. adikari, "Area Efficient Multipliers Based on Multiple-Radix Representations," IEEE Trans. on Computers, vol. 60, N° 2, pp 189-201, February 2011

[13] O.L. McSorley, "High-Speed Arithmetic in Binary Computers," Proceedings of the IRE, Vol. 49(1), pp. 67-91, January 1961.

[14] F. Lamberti, "Reducing the Computation Time in (Short Bit-Width) Two's Complement Multiplier," IEEE Trans. on Computers, vol. 60, N° 2, pp. 148-156, February 2011.

[15] S.R. Kuang, J.P. Wang, and C.Y. Guo, "Modified Booth Multipliers with a Regular Partial Product Array," IEEE Trans. on Circuit and Systems II, Express Brief, vol. 56, N° 5, May 2009.

[16] S.R. Kuang, J.P. Wang, "Design of Power-Efficient Configurable Booth Multiplier," IEEE Trans. on Circuit and Systems I, vol. 57, N° 3, March 2010.

[17] M. Själander and P. Larsson-Edefors, "Multiplication Acceleration Through Twin Precision," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 17, N° 9, September 2009.

[18] A. D. Booth, "A Signed Binary Multiplication Technique," Quarterly J. Mech. Appl. Math., Vol. 4, part 2, pp. 236-240,1951.

[19] P.M. Seidel, L. D. McFearin, and D.W. Matula, "Binary Multiplication Radix-32 and Radix-256," Proceedings of the IEEE Symposium on Computer Arithmetic (ARITH-15), ISBN: 0-7695-1150-3, pp. 23-32, USA, June 2001.

[20] A.K. Oudjida et *al*., "A New Recursive Multibit Recoding Algorithm for High-Speed and Low-Power Multiplier," Accepted for publication in Journal of Low Power Electronics (JOLPE), Vol. 8, N° 5, December 2012.

[21] E. Manmasson et *al*., "FPGA in Industrial Control Applications," IEEE Trans. on Industrial Informatics, vol. 7, N° 2, May 2011.

[22] M. Alioto, Elio Consoli, and Gaetano Palumbo, "Metrics and Design Consideration on the Energy-Delay Tradoff of Digital Circuits," Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'09), pp. 3150-3153, Taiwan, May 24-27 2009.

[23] A.K. Oudjida et *al*., "High-Speed and Low-Power PID Structures for Embedded Applications," Proceedings of the 21th edition of the International Workshop on Power and Timing Modeling, Optimization and Simulation PATMOS, LNCS 6951, pp. 257-266, Springer-Verlag Editor. Madrid, Spain, September 26-29, 2011.

APPENDIX

Proof of theorem 1: Initially, the multiplier Y is an N bit string. But to comply with the requirement of the multibit recoding algorithm [2], we need to add a zero bit (y_{-1}) to the less significant side of Y. Thus, the total size becomes N+1.

$$Y = y_{-1}\ y_0\ y_1\ y_2 \cdots y_{N-2}\ y_{N-1} \qquad \text{with} \quad y_{-1}=0$$

Y is a two's complement number. It is written as follows:

$$\begin{aligned} Y &= y_{-1} + 2^0 y_0 + 2^1 y_1 + 2^2 y_2 + \cdots + 2^{N-2} y_{N-2} - 2^{N-1} y_{N-1} \\ &= -y_{N-1}\, 2^{N-1} + \sum_{j=-1}^{N-2} y_j\, 2^j \end{aligned}$$

In the multibit recoding algorithm, the multiplier Y is split into N/r two's complement slices (Q_j), each of r+1 bit length. Two contiguous slices (Q_j with Q_{j-1}, and Q_j with Q_{j+1}) have one overlapping bit in common. Thus Y becomes:

$$Y = \sum_{j=0}^{\frac{N}{r}-1} \left(y_{rj-1} + 2^0 y_{rj} + 2^1 y_{rj+1} + 2^2 y_{rj+2} + \cdots + 2^{r-2} y_{rj+r-2} - 2^{r-1} y_{rj+r-1} \right) 2^{rj} = \sum_{j=0}^{\frac{N}{r}-1} Q_j\, 2^{rj}$$

In fact the Q_j term is no more than a two's complement representation of $r+1$ bit string which can be split in its turn into r/s two's complement overlapping slices (P_{ji}), each of s+1 bit length. Thus Y becomes:

$$\begin{aligned} Y = \sum_{j=0}^{\frac{N}{r}-1} \Big[& \left(y_{rj-1} + 2^0 y_{rj} + 2^1 . y_{rj+1} + 2^2 y_{rj+2} + \cdots + 2^{s-2} y_{rj+s-2} - 2^{s-1} y_{rj+s-1} \right) 2^0 \quad + \\ & \left(y_{rj+s-1} + 2^0 y_{rj+s} + 2^1 . y_{rj+s+1} + 2^2 y_{rj+s+2} + \cdots + 2^{s-2} y_{rj+2s-2} - 2^{s-1} y_{rj+2s-1} \right) 2^s \quad + \\ & \qquad \vdots \\ & \left(y_{rj+r-1-2s} + 2^0 y_{rj+r-2s} + 2^1 . y_{rj+r-2s+1} + 2^2 y_{rj+r-2s+2} + \cdots + 2^{s-2} y_{rj+r-s-2} - 2^{s-1} y_{rj+r-1-s} \right) 2^{s\left(\frac{r}{s}-2\right)} \quad + \\ & \left(y_{rj+r-1-s} + 2^0 y_{rj+r-s} + 2^1 . y_{rj+r+s+1} + 2^2 y_{rj+r+s+2} + \cdots + 2^{s-2} y_{rj+r-2} - 2^{s-1} y_{rj+r-1} \right) 2^{s\left(\frac{r}{s}-1\right)} \Big] \end{aligned}$$

$$= \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s}-1} \left(y_{rj-1+si} + 2^0 y_{rj+si} + 2^1 y_{rj+1+si} + 2^2 y_{rj+2+si} + \cdots + 2^{s-2} y_{rj+s-2+si} - 2^{s-1} . y_{rj+s-1+si} \right) 2^{si} \right] 2^{rj}$$

$$= \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s}-1} P_{ji} \, 2^{si} \right] 2^{rj}$$

A synoptic scheme is depicted in Fig. 1 to illustrate the use of theorem 1 in the partitioning of a 16-bit *Y* operand.

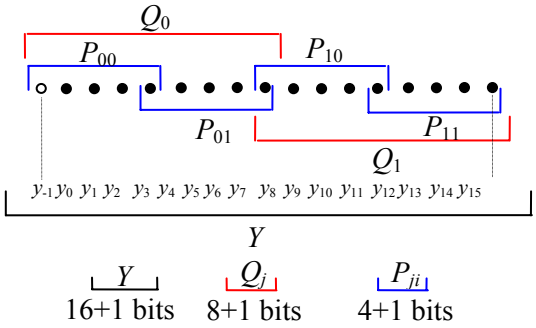

Figure 1. Partitioning of a 16-bit Y operand with r=8 and s=4

Proof of theorem 2: Likewise, *Y* can also be rewritten as follows:

$$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s+t}-1} \left[\left(y_{rj-1+(s+t)i} + 2^0 y_{rj+(s+t)i} + 2^1 . y_{rj+1+(s+t)i} + \ldots + 2^{s-2} y_{rj+s-2+(s+t)i} - 2^{s-1} y_{rj+s-1+(s+t)i} \right) + \right. \right.$$
$$\left. \left. \left(y_{rj+s-1+(s+t)i} + 2^0 y_{rj+s+(s+t)i} + 2^1 . y_{rj+1+s+(s+t)i} + \cdots + 2^{t-2} y_{rj+r-2+(s+t)i} - 2^{t-1} y_{rj+r-1+(s+t)i} \right) 2^s \right] 2^{(s+t)i} \right] 2^{rj}$$

$$= \sum_{j}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s+t}-1} \left[P_{ji} + T_{ji} \, 2^s \right] 2^{(s+t)i} \right] 2^{rj}$$

A synoptic scheme is depicted in Fig. 2 to illustrate the use of theorem 2 in the partitioning of a 16-bit *Y* operand.

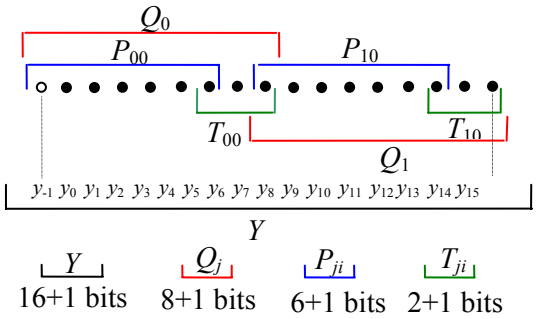

Figure 2. Partitioning of a 16-bit Y operand with r=8, s=6 and t=2